apresentação.py me.py

```python
Frasko = (
    ‘Desenvolvendo seu próprio’
    ‘nano web framework’
) # Versão "Se vira nos 30"
```

Geraldo Castro - Python Floripa 27/agosto/2022 apresentação.py me.py

```python
geraldo_castro = [

    ‘Mossoró/RN → Florianópolis/SC’,
    ‘backend na maioria do tempo’,
    ‘diabético’,
    ‘vegetariano’,
    ‘amo camisas de evento’,
    ‘amo eventos’,

]
```

me.py qrcode.png

```
# https://bit.ly/exageraldo-na-pythonfloripa-65
```

qrcode.png sumário.py

```
conteúdo = {

    00:  (razões and considerações and referências),
    01:  ((web_app and web_server) and wsgi),
    02:  (dependências),
    03:  (request and response),
    04:  ((rotas_simples and rotas_parametrizadas) or 404),
    05:  (class_based_decorators and rotas_duplicadas),

}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14

```python
# app
from frasko import Frasko, Request, Response

app = Frasko()

@app.route("/home")
def index(request: 'Request', response: 'Response') -> None:
    response.text = "um OLAR do index (GET)!"

@app.route("/user", method="post")
def sobre(request: 'Request', response: 'Response') -> None:
    response.text = "um OLAR do sobre (POST)!"

@app.route("/olar/{vezes:d}")
def olar_x_vezes(
    request: 'Request',
    response: 'Response',
    vezes: int,
) -> None:
    response.text = f"{'OLAR ' * vezes} (GET)"

@app.route("/olar/{nome:w}")
def olar_fulano(
    request: 'Request',
    response: 'Response',
    nome: str,
) -> None:
    response.text = f"OLAR {nome} (GET)"

@app.route("/sobremesas")
class BooksResource:
    def get(self, request: 'Request', response: 'Response') -> None:
        response.text = "um OLAR de sobremesas (GET)!"

    def post(self, request: 'Request', response: 'Response') -> None:
        response.text = "um OLAR de sobremesas (POST)!"
```

prévia.py razões_considerações_referências.py

```
(

    razões and
    considerações and
    referências
)
                # conteúdo[00]
```

razões_considerações_referências.py razões.py

```
por_que_criar_um_framework = [
    'por que criar meu próprio _____ ?',
    'por razões de aprendizado/estudo', # nosso caso
    'por precisar de algo muito específico',
    'por que não?',
]
```

razões.py considerações.py

```python
considerações_iniciais = [

    'familiaridade com python',

    'conhecimento mínimo sobre web',

    'já ter utilizado algum framework web',

    'nano < micro', # Frasko < Flask

]
```

considerações.py   referências.py

```
referências = {

    ‘How to write a Python web framework (free/blog post version)’: {
        ‘autor’: ‘Jahongir Rahmonov’,
        ‘link’: link_um,
    },
    ‘How to write a Python web framework (paid/testdriven.io version)’: {
        ‘autor’: ‘Jahongir Rahmonov’,
        ‘link’: link_dois,
    },
...
```

considerações.py   referências.py

```
...
    ‘Let's build a web framework! PyCon 2017’: {
        ‘autor’: ‘Jacob Kaplan Moss’,
        ‘link’: link_tres,
    },
    ‘Let’s Build A Web Server [Part 2]’: {
        ‘autor’: ‘Ruslan Spivak’,
        ‘link’: link_quatro,
    },
...
```

considerações.py referências.py

```
...
    ‘WGSI Tutorial’: {
        ‘autor’: ‘Clodoaldo Pinto Neto’,
        ‘link’: link_cinco,
    },
    ‘[EXTRA] Build Your Own X’: {
        ‘autor’: ‘comunidade/open source’,
        ‘link’: link_seis,
    },
}
```

```
(
    (web_server and web_app)
    and wsgi
)

        # conteúdo[01]
```

server_app_wsgi.py   server.py

```
web_server = [

    ‘espera pacientemente por uma requisição (Request)’,
    ‘recebe um request do cliente e envia para um "PythonApp"’,
    ‘espera pelo processamento da resposta (Response)’,
    ‘envia a resposta para o cliente de volta’,

] # exemplos: gunicorn, uwsgi
```

server.py app.py

```
web_app = [

    'recebe a requisição enviada pelo web server',
    'executa alguns comandos definido em regras definidas',
    'monta a resposta e devolve para o web server',

] # exemplos: flask, django, bottle
```

app.py wsgi.py

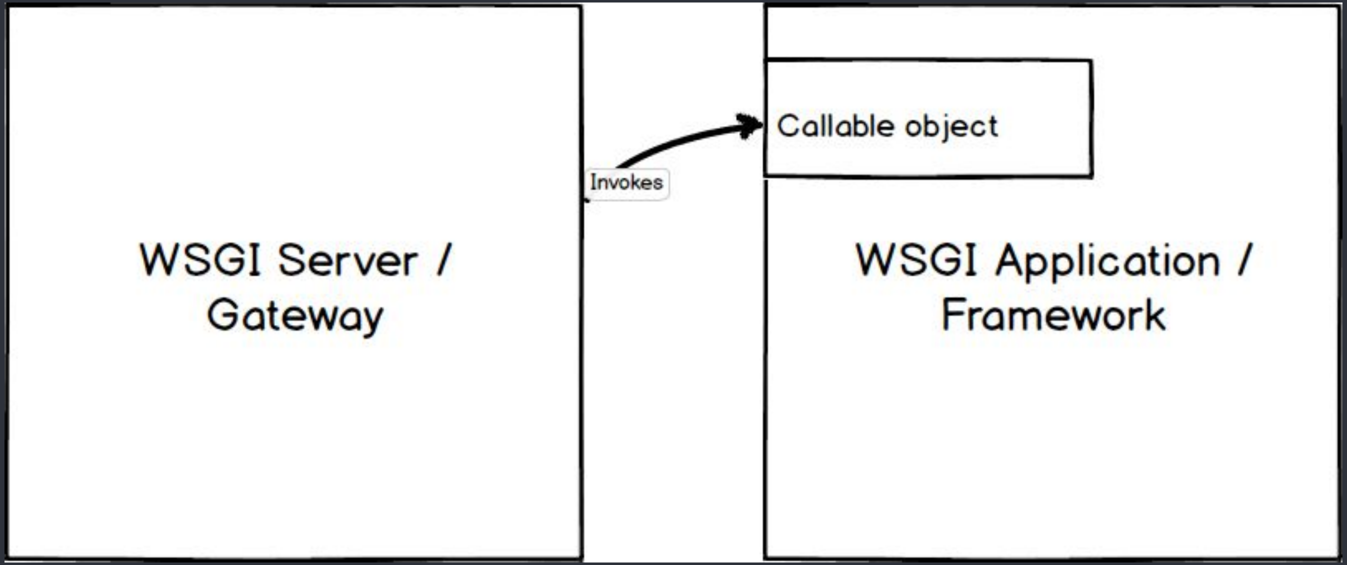
```
problema = [
    ‘quem desenvolve o app não quer lidar com o server (vice versa)’,
    ‘incompatibilidade entre web app e web server → limitação’,
    ‘criar adaptadores entre o app e o server (mod_python - Apache)’,
]
```

app.py wsgi.py

```
wsgi = [

    ‘Web Server Gateway Interface’,

    ‘não é um servidor, um módulo python, um framework, uma API ou
    qualquer tipo de software’,

    ‘é uma especificação de comunicação entre o servidor e a
    aplicação’,

    ‘ambos os lados devem aplicar as especificações’,

    ‘PEP 3333’, # link
...
```

wsgi.py

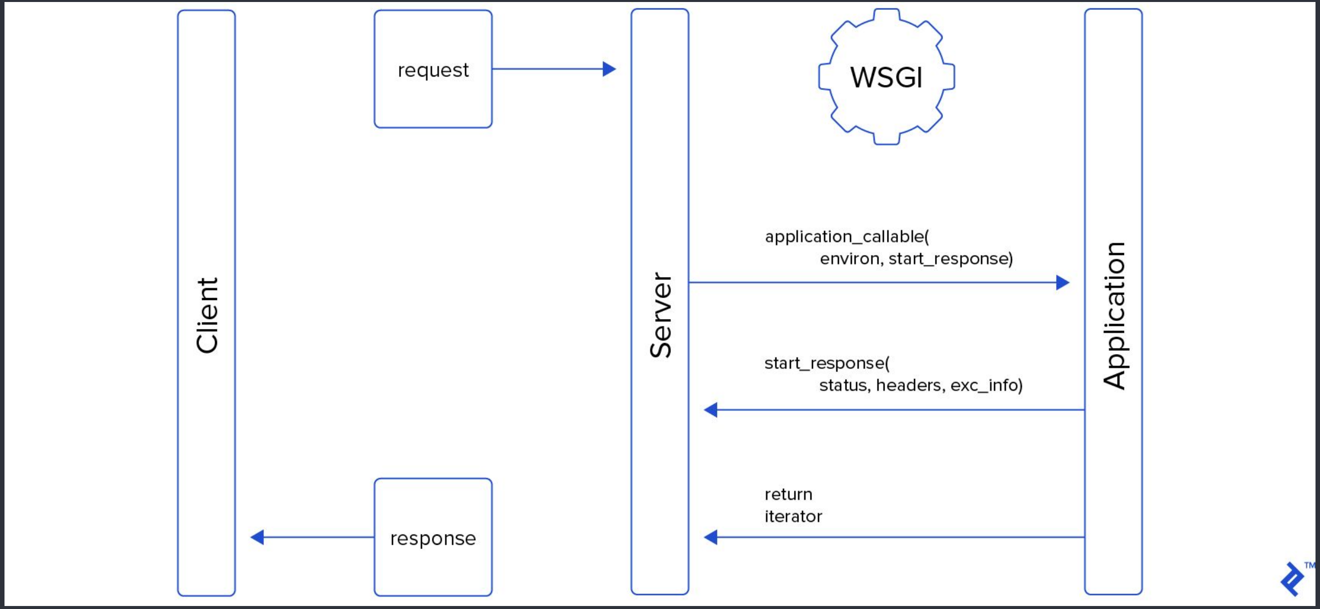
wsgi_um.png



wsgi_um.png wsgi.py

```
...
    ‘server: deve chamar o objeto app com os parâmetros environ e
    start_response ("application(environ, start_response)")’,

    ‘app: deve chamar a função start_response com o status_code e
    headers_response ("start_response(status_code, headers_response)")
    antes de retornar o body para o server’,

    ‘o body deve ser um interavel (Iterable)’,
]
```

wsgi.py wsgi_dois.png



```
(
    dependências
)

            # conteúdo[02]
```

dependências.py requirements.txt

```
# desenvolvimento
webob # Request & Response wrapper
parse # ajuda na parametrização das rotas

# testar/rodar
gunicorn
```

requirements.txt request_response.py

(request and response)

# conteúdo[03]

request_response.py

app.py

```python
# app.py
from frasko import frasko

# apenas pra manter o mesmo formato
app = frasko
```

# link do código

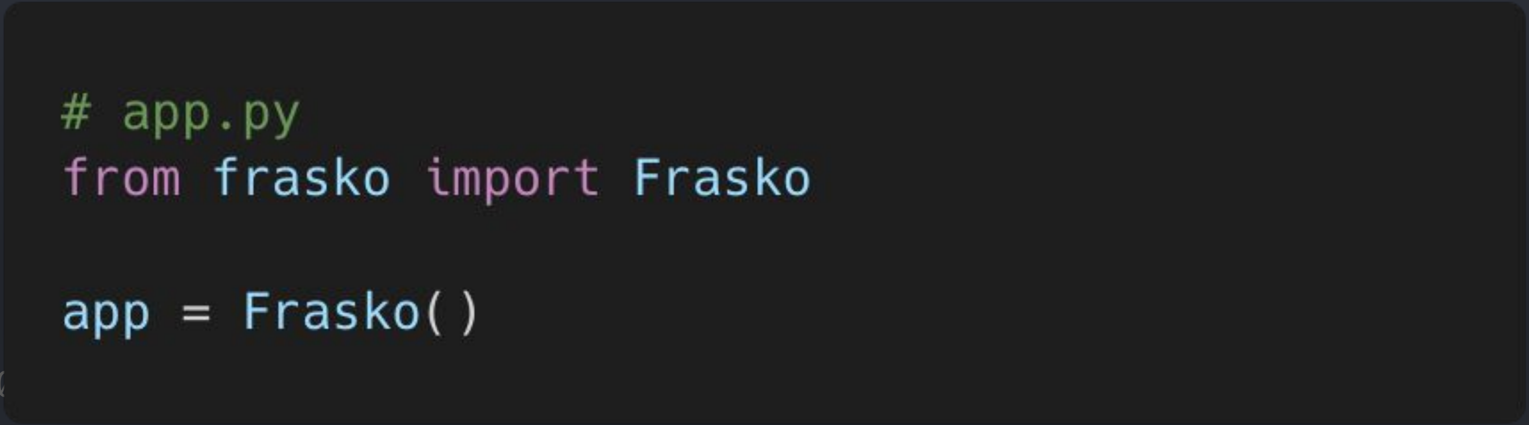
```python
# frasko.py
from typing import Dict, Callable, List, Optional, Tuple
from webob import Response, Request

def frasko(
    environ: 'Dict',
    start_response: 'Callable[[str, List, Optional[Tuple]], Callable]',
):
    '''
    environ: the environ dictionary is required to contain these CGI environment variables
    start_response: start_response(status, response_headers, exc_info=None) -> write(text_in_bytes)
    '''

    request = Request(environ) # vamos usar jaja
    response = Response()
    response.text = "passo 00"
    response.status_code = 200

    return response(environ, start_response)
```

# link do código

frasko.py   environ.py

```python
sample_environ = {
    'HTTP_ACCEPT': '*/*',
    'HTTP_ACCEPT_ENCODING': 'gzip, deflate, br',
    'HTTP_CONNECTION': 'close',
    'HTTP_HOST': '127.0.0.1:8000',
    'HTTP_USER_AGENT': 'Thunder Client (https://www.thunderclient.com)',
    'PATH_INFO': '/',
    'QUERY_STRING': '',
    'RAW_URI': '/',
    'REMOTE_ADDR': '127.0.0.1',
    'REMOTE_PORT': '59398',
    'REQUEST_METHOD': 'GET',
    'SCRIPT_NAME': '',
    'SERVER_NAME': '127.0.0.1',
    'SERVER_PORT': '8000',
    'SERVER_PROTOCOL': 'HTTP/1.1',
    'SERVER_SOFTWARE': 'gunicorn/20.0.4',
    'gunicorn.socket': <socket.socket fd=9, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 
8000), raddr=('127.0.0.1', 59398)>,
    'wsgi.errors': <gunicorn.http.wsgi.WSGIErrorsWrapper object at 0x10e0369b0>,
    'wsgi.file_wrapper': <class 'gunicorn.http.wsgi.FileWrapper'>,
    'wsgi.input': <gunicorn.http.body.Body object at 0x10e0369b0>,
    'wsgi.input_terminated': True,
    'wsgi.multiprocess': False,
    'wsgi.multithread': False,
    'wsgi.run_once': False,
    'wsgi.url_scheme': 'http',
    'wsgi.version': (1, 0)
}
```

# melhorando nossa interface



```python
# app.py
from frasko import Frasko

app = Frasko()
```

app.py frasko.py

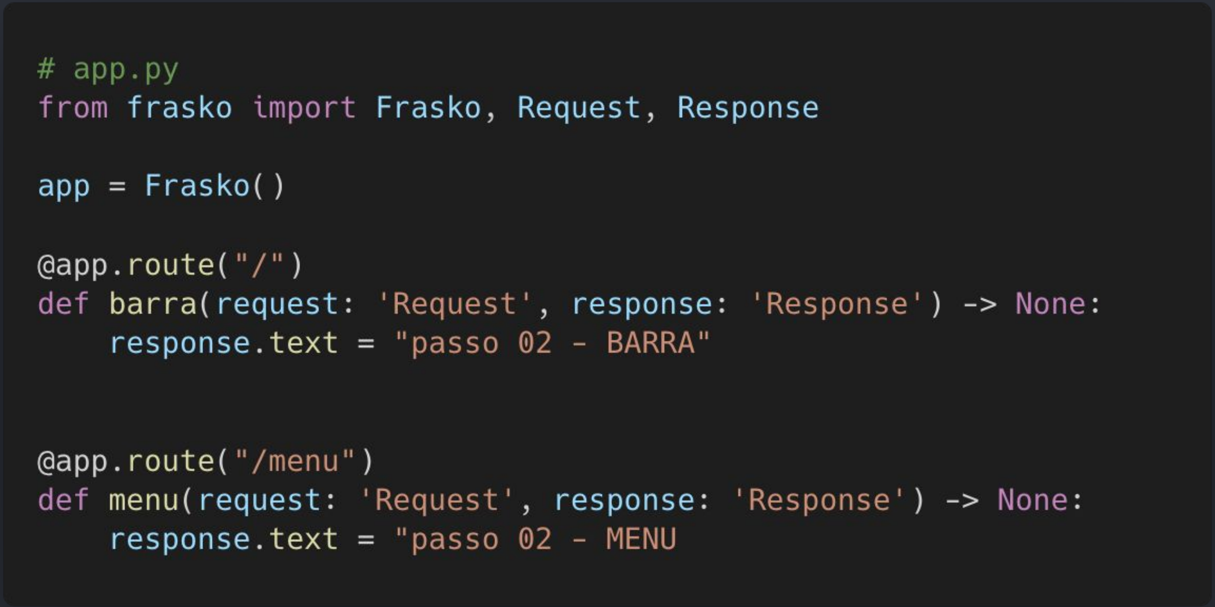
```python
# frasko.py
from typing import Dict, Callable, List, Optional, Tuple
from webob import Response, Request

class Frasko:
    def __call__(
        self,
        environ: 'Dict',
        start_response: 'Callable[[str, List, Optional[Tuple]], Callable]',
    ):
        request = Request(environ)
        response = self._handle_request(request)

        return response(environ, start_response)

    def _handle_request(self, request: 'Request') -> 'Response':
        response = Response("passo 01", 200)

        return response
```

# link do código

frasko.py   rotas_simples_parametrizadas.py

```
(
    rotas_simples and
    rotas_parametrizadas
)
          # conteúdo[04]
```

# rotas simples



```python
# app.py
from frasko import Frasko, Request, Response

app = Frasko()

@app.route("/")
def barra(request: 'Request', response: 'Response') -> None:
    response.text = "passo 02 - BARRA"


@app.route("/menu")
def menu(request: 'Request', response: 'Response') -> None:
    response.text = "passo 02 - MENU
```

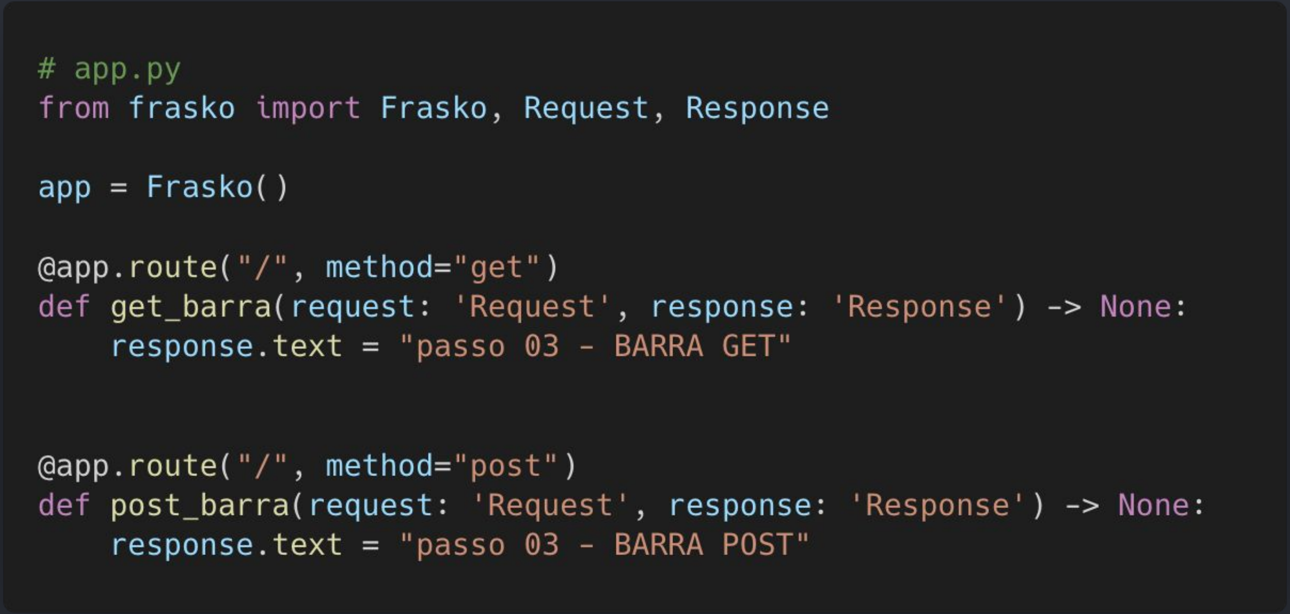
app.py frasko.py

```python
# frasko.py
from typing import Dict, Callable, List, Optional, Tuple
from webob import Response, Request

class Frasko:
    def __init__(self) -> None:
        self._routes = {}

    def __call__(
        self,
        environ: 'Dict',
        start_response: 'Callable[[str, List, Optional[Tuple]], Callable]',
    ):
        request = Request(environ)
        response = self._handle_request(request)

        return response(environ, start_response)

    def route(self, path: str):
        def wrapper(handler: 'Callable[[Request, Response], None]'):
            self._routes[path] = handler
            return handler

        return wrapper

    def _handle_request(self, request: 'Request') -> 'Response':
        response = Response()

        for path, handler in self._routes.items():
            if path == request.path:
                handler(request, response)
                return response

        return response
```

# link do código

frasko.py app.py

# definindo rotas com verbos (explícitos)



```python
# app.py
from frasko import Frasko, Request, Response

app = Frasko()

@app.route("/", method="get")
def get_barra(request: 'Request', response: 'Response') -> None:
    response.text = "passo 03 - BARRA GET"


@app.route("/", method="post")
def post_barra(request: 'Request', response: 'Response') -> None:
    response.text = "passo 03 - BARRA POST"
```

# link do código

app.py frasko.py

```python
# frasko.py
from typing import Dict, Callable, List, Optional, Tuple
from webob import Response, Request

class Frasko:
    def __init__(self) -> None:
        self._routes = {}

    def __call__(
        self,
        environ: 'Dict',
        start_response: 'Callable[[str, List, Optional[Tuple]], Callable]',
    ):
        request = Request(environ)
        response = self._handle_request(request)

        return response(environ, start_response)

    def route(self, path: str, method="get"):
        routes = self._routes.setdefault(method, {})
        def wrapper(handler: 'Callable[[Request, Response], None]'):
            routes[path] = handler
            return handler

        return wrapper

    def _handle_request(self, request: 'Request') -> 'Response':
        response = Response()
        routes = self._routes.get(request.method.lower(), {})
        for path, handler in routes.items():
            if path == request.path:
                handler(request, response)
                return response

        return response
```

# link do código

frasko.py app.py

# parametrizando as rotas

```python
# app.py
from frasko import Frasko, Request, Response

app = Frasko()

@app.route("/olar/{vezes:d}")
def olar_x_vezes(request: 'Request', response: 'Response', vezes: int) -> 
None:
    response.text = f"passo 05 - GET {'OLAR ' * vezes}"

@app.route("/olar/{nome:w}")
def olar_fulano(request: 'Request', response: 'Response', nome: str) -> 
None:
    response.text = f"passo 05 - OLAR {nome} GET"
```

# link do código

app.py

frasko.py

```python
# frasko.py
from typing import Dict, Callable, List, Optional, Tuple
from webob import Response, Request
from parse import parse

class Frasko:
    def __init__(self) -> None:
        self._routes = {}

    def __call__(
        self,
        environ: 'Dict',
        start_response: 'Callable[[str, List, Optional[Tuple]], Callable]',
    ):
        request = Request(environ)
        response = self._handle_request(request)

        return response(environ, start_response)

    def route(self, path: str, method="get"):
        routes = self._routes.setdefault(method, {})
        def wrapper(handler: 'Callable[[Request, Response], None]'):
            routes[path] = handler
            return handler

        return wrapper
...
```

```python
...

    def _default_response(self, response: 'Response') -> None:
        response.text = "NOT FOUND"
        response.status_code = 404

    def _handle_request(self, request: 'Request') -> 'Response':
        response = Response()
        routes = self._routes.get(request.method.lower(), {})
        for path, handler in routes.items():
            parse_result = parse(path, request.path)
            if parse_result is None:
                continue

            handler(request, response, **parse_result.named)
            return response

        self._default_response(response)
        return response
```

# link do código

frasko.py class_decorators_rotas_duplicadas.py

```
(
    class_based_decorators and
    rotas_duplicadas
)
            # conteúdo[05]
```

# decorando classes

```python
# app.py
from frasko import Frasko, Request, Response

app = Frasko()

@app.route("/book")
class BooksResource:
    def get(self, request: 'Request', response: 'Response'):
        response.text = "passo 06 - BOOK GET"

    def post(self, request: 'Request', response: 'Response'):
        response.text = "passo 06 - BOOK POST"
```

app.py

frasko.py

```python
# frasko.py
from typing import Dict, Callable, List, Optional, Tuple
import inspect
from webob import Response, Request
from parse import parse

class FraskoException(Exception):
    """ A base class for exceptions used by frasko. """
    pass

class Frasko:
    def __init__(self) -> None:
        self._routes = {}

    def __call__(
        self,
        environ: 'Dict',
        start_response: 'Callable[[str, List, Optional[Tuple]], Callable]',
    ):
        request = Request(environ)
        response = self._handle_request(request)

        return response(environ, start_response)

    def _default_response(self, response: 'Response') -> None:
        response.text = "NOT FOUND"
        response.status_code = 404

    def route(self, path, method="get"):
        def wrapper(handler):
            self.add_route(path, handler, method)
            return handler
        return wrapper
...
```

```python
...

    def add_route(self, path, handler, method="get"):
        if inspect.isclass(handler):
            methods = set(vars(handler).keys()) & set(
                ["get", "post", "put", "delete"]
            )
            for method in methods:
                routes = self._routes.setdefault(method, {})
                if path in routes:
                    raise FraskoException("Such route already exists.")
                routes[path] = getattr(handler(), method)
        else:
            routes = self._routes.setdefault(method, {})
            if path in routes:
                raise FraskoException("Such route already exists.")
            routes[path] = handler

    def _handle_request(self, request: 'Request') -> 'Response':
        response = Response()
        routes = self._routes.get(request.method.lower(), {})
        for path, handler in routes.items():
            parse_result = parse(path, request.path)
            if parse_result is None:
                continue

            handler(request, response, **parse_result.named)
            return response

        self._default_response(response)
        return response
```

# link do código

frasko.py

obrigado_perguntas.py

(

obrigado and
perguntas

)

# https://bit.ly/exageraldo-na-pythonfloripa-65

Slide Template: Slidesgo.com